Sqirvy-AI : Command Line Agents

VERSION 0.0.1-alpha

Table of Contents

What If You Could String Together Some AI Queries To Make Something Happen? And Use A Differet LLM For Each Step? And Do It From The Terminal Instead Of A UI?

Imagine you are setting up some DevOps for a project, and you need a simple way to make queries to LLM providers for use in a command line program. You don't want to have to copypasta from a web app or a python script. Or, you want to automate tasks like code review or web scraping using LLMs.

How about this: have a set of simple command line programs that perform various fixed queries to LLM providers. You can use them to automate tasks like code review, testing, and deployment. They could be used in CI/CD pipelines, or as part of a devops workflow. And each step could use a different LLM, whatever was suitable for the options.

What if you could chain multiple queries in a shell command or script, and get a single response?

That's what this project is all about.

Note: Each LLM model will give different results for a given prompt, and each execution of the same program and prompt will most likely generate different results even with the same model and prompt.

GitHub Repo

LLMs Supported In This Release

- https://www.anthropic.com/api
  - claude-3-5-sonnet-latest
  - claude-3-5-haiku-latest
  - claude-3-opus-latest
- https://ai.google.dev/gemini-api/docs
  - gemini-2.0-flash-exp
  - gemini-1.5-flash
  - gemini-1.5-pro
- https://platform.openai.com/docs/overview
  - gpt-4o
  - gpt-4o-mini
  - gpt-4-turbo
  - o1-mini
- https://docs.llama-api.com/quickstart
- llama3.3-70b
- deepseek-r1 (tested with Meta Llama provider)

How It Works

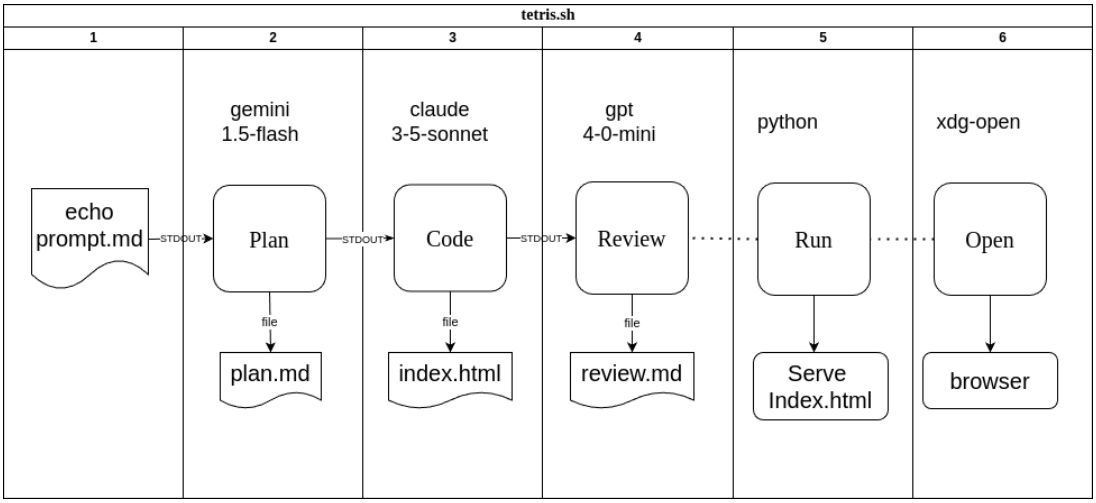
The main program is **sqirvy**, which is a command line utility that can perform AI queries. **sqirvy** is setup to take input from stdin, perform a query to a specified LLM and send the results to stdout. Because it uses **stdin | sqirvy | stdout**, it is possible to chain together a pipeline, using the same of different models and LLM providers at each step. In cases where a query needs multiple inputs, it supports taking file names and urls as arguments.

```
Usage: bin/sqirvy [-h] [-m model] [-f function] [-t temperature] [files
and/or urls  ...]
  -h  print this help message
  -m  AI model to use (default: claude-3.5-sonnet-latest)
  -f  AI function to use (default: query)
    -f query  : execute a generic query
    -f plan   : generate a plan for further action
    -f code   : generate code according to input specifications
    -f review : review code or text
    -f scrape : scrape a URL for text
  -t  Temperature setting for the AI model (0-100, default: 50)
  file1 file2 ...  input files to process
  URLs can be provided as arguments
  data from stdin will be read if there is any
```

Example Scripts

There are example bash scripts that illustrate the type of actions you can take with the **sqirvy** program to perform multi-step operations in a pipeline.

scripts/tetris




```
#!/bin/bash

# in scripts/tetris
# this script does the following:
# - creates a directory called tetris
# - echo the design prompt into stdin of sqirvy in plan mode.
#   - pipe the plan to stdout
# - pipe stdin to sqirvy -f plan and gemini-1.5-flash to create a design
for a web app
#   - tee the plan to stdout and to a file
# - pipe stdin to sqirvy -f code and claude-3-5-sonnet-latest to generate
code for the design
#   - tee the code to stdout and to a file
# - pipe stin to sqirvy -f review and gpt-4o-mini to review the code
# - starts a web server to serve the generated code

design_prompt="create a design specification for a web project that is a \
    simple web app that implements a simple tetris game clone.       \
    the game should include a game board with a grid, a score display, and
a reset button \
    Code should be html, css and javascript, in a single file named
index.html. \
    Output will be markdown.  "

export BINDIR=../bin
make -C ../cmd

rm -rf tetris && mkdir tetris
echo $design_prompt | \
$BINDIR/sqirvy -m gemini-1.5-flash         -f plan    | tee tetris/plan.md
| \
$BINDIR/sqirvy -m claude-3-5-sonnet-latest -f code    | tee
tetris/index.html | \
$BINDIR/sqirvy -m gpt-4o-mini              -f review  >tetris/review.md

python -m http.server 8080 --directory tetris &
```

```
# for Ubuntu. for other platforms just open a browser to this url
xdg-open http://localhost:8080
```

Sqirvy-llm Command Line Programs

Supported Models

- Supported models:
    - Anthropic
        - claude-3-5-haiku-latest
        - claude-3-5-sonnet-latest
        - claude-3-opus-latest
    - Google Gemini
        - gemini-1.5-flash
        - gemini-1.5-pro
        - gemini-2.0-flash-exp
    - OpenAI
        - gpt-4-turbo
        - gpt-4o
        - gpt-4o-mini
        - o1-mini
    - Meta-Llama
        - meta-llama/meta-llama-3.1-8b-instruct-turbo
        - meta-llama/Llama-3.3-70B-Instruct-Turbo
        - meta-llama/Meta-Llama-3.1-405B-Instruct-Turbo
    - DeepSeek
    - deepseek-r1
    - deepseek-chat (also disabled)

sqirvy-query

- run an arbitrary query to an LLM provider
- concatenates prompt from stdin and/or files and sends it to the specified AI model
- defaults to Anthropic claude-3-5-sonnet-latest model if no model is specified
- example: pipe a prompt to sqirvy-query with the default claude-3-5-sonnet-latest model
    - echo "say hello world" | sqirvy-query
- example: read a prompt from a file and pipe it to sqirvy-query with the o1-mini model
    - sqirvy-query -m o1-mini prompt.txt

A command line program that allows you to send arbitrary prompts to an AI model.

Usage: sqirvy-query [options] files...

- Options:
  - -h print this help message
  - -m AI model to use (default: claude-3-5-sonnet-latest)

Example usage:

```
sqirvy-query -m gpt-4-turbo-preview "tell me a joke"

sqirvy-query -h
Usage: sqirvy-query [options] files...
initializes the context from stdin, pipe or redirection (if any)
concatenates files to the context in order
Options:
  -h    print this help message
  -m    AI model to use (default: claude-3-5-sonnet-latest)
```

sqirvy-review

A command line program that invokes an AI model to perform code review. This program has a built-in system and review prompt (using Go file embeddeding), so you don't need to provide any prompts if you don't want to. If you don't like those prompts, you can modify them and rebuild the program.

Example usage:

```
sqirvy-review -m claude-3-5-haiku-latest ../../cmd/sqirvy-query/*.go

sqirvy-review -h
Usage: sqirvy-review [options] files...
initializes the context from stdin, pipe or redirection (if any)
concatenates files to the context in order
Options:
  -h    print this help message
  -m    AI model to use (default: gemini-1.5-flash)
```

sqirvy-scrape

A command line program that invokes an AI model to scrape data from the web and perform some action on the downloaded data.

Example usage:

```
sqirvy-scrape -m gpt-4-turbo-preview https://sqirvy.xyz

sqirvy-scrape -h
Usage: sqirvy-scrape [options] urls...
initializes the context from stdin, pipe or redirection (if any)
scrapes content from URLs and sends it to the specified AI model
Options:
  -h     print this help message
  -m     AI model to use (default: claude-3-5-sonnet-latest)
```

Chaining

The command line programs can be chained together to perform more complex tasks. This is because the prompt inputs to the programs are added to the context in this order:

- special cases
    - sqirvy-review has a built-in system and query prompt.
    - sqirvy_query and sqirvy_scrape, if a file named system.md exists in the current directory, it will be first in the context.
- stdin, pipe or redirection.
- files specified on the command line

For example, you can use sqirvy-scrape to scrape a website and then use sqirvy-query to summarize the content.

```
sqirvy-scrape -m gpt-4-turbo-preview https://sqirvy.xyz | sqirvy-query -m
gpt-4-turbo-preview "summarize the content"
```

SDK Library

The above preconfigure commands use the sqirvy-llm/pkg/sqirvy SDK in this repo. This is the interface you would use to make queries to LLM providers in Go if you want to use it in your own Go programs.

Most of the code was generated using Aider and the claude-3-sonnet-20240229 model. I had to do several iterations with Aider and some manual editing to get the exact code layout I wanted.

The API is in directory pkg/sqirvy. It is a very simple interface that allows you to query a provider with a prompt and get a response. It supports Anthropic, Gemini, and OpenAI providers through the 'client' interface. Here is an example of how to use the API in a command line program. Examples for the other providers are in the 'cmd' directory.

- Making a query to a provider
    - Create a new client for the provider you want to use
        - sqirvy.NewClient(sqirvy.)
        - anthropic, gemini or openai

    - Make the query with a prompt, the model name, and any options (nothing supported yet). You can request the results to be plain text or JSON
        - client.QueryText(prompt, model string, options Options) (string, error)
        - client.QueryJSON(prompt string, model string, options Options) (string, error)
    - Get the response
    - Handle any errors

```
package main

import (
    "fmt"
    "log"

    sqirvy "sqirvy-llm/pkg/sqirvy"
)

func main() {
    // Create a new Anthropic client
    client, err := sqirvy.NewClient(sqirvy.Anthropic)
    if err != nil {
        log.Fatalf("Failed to create client: %v", err)
    }

    // Make the query with a prompt, the model name, and any options
(nothing supported yet)
    response, err := client.QueryText("say hello world", "claude-3-sonnet-
20240229", sqirvy.Options{})
    if err != nil {
        log.Fatalf("Query failed: %v", err)
    }

    fmt.Println("Response:", response)
}
```

Example Usage

Example code is in directory **examples**. To use them, first build the binaries.

Build The Executables

- the build system uses GNU 'make'
- 'make' can be run from top level or from the cmd or web directories
- build (or default)
    - build the binaries for the cmd and web directories
    - it attempts to build the version for the current OS and CPU Architecture. Has been tested on Ubuntu Linux and Windows 11. Not tested on MacOS x86 or Apple silicon
    - after a `make` or `make test`, the binaries will be in the top level `bin` directory
- test
    - run the tests

- clean
  - remove the binaries an clanup temporary files

Examples

Simple hard coded queries using the specified provider:

- examples/anthropic
- examples/gemini
- examples/meta-llama
- examples/openai

Examples for the various -f flags

- examples/sqirvy-query : generic chat query
- examples/sqirvy-plan : generate a plan for an application
- examples/sqirvy-code : generate code for an application
- examples/sqirvy-review : review existing code
- examples/sqirvy-scrape : scrape a web page and summarize it

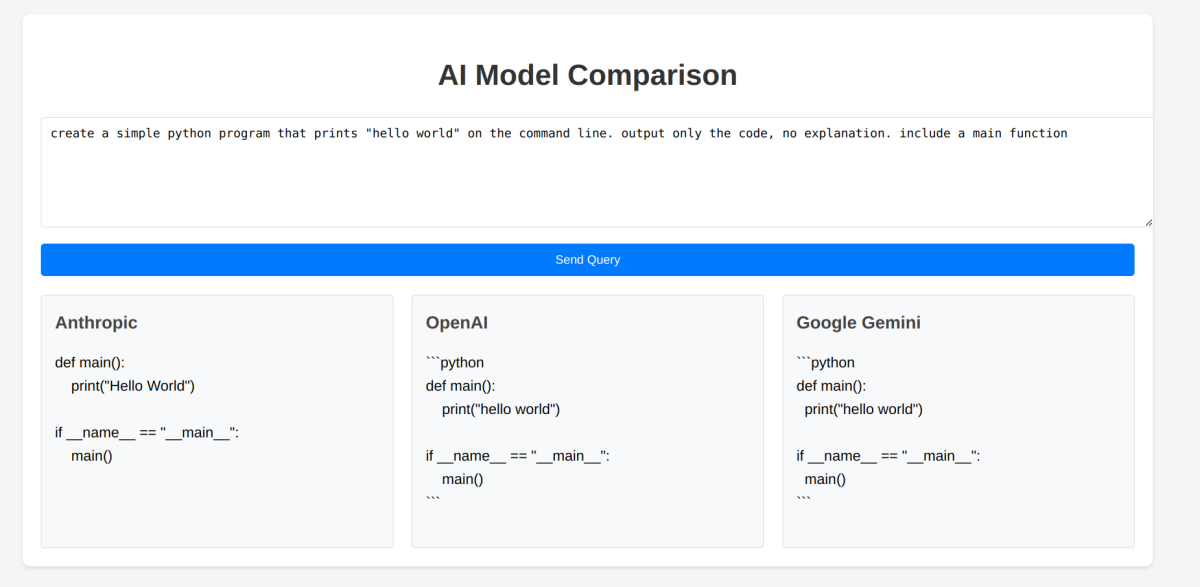
Example web app

- examples/web : for comparing results of queries from 3 different models

web/sqirvy-web

A simple web app that allows you to query all three providers in parallel and compare the results.

- cd into "web/sqirvy-web"
- go run .
- it will start a web server on port 8080.



The code for the web app was generated using Aider and the claude-3-sonnet-20240229 model.

CLients

Anthropic

- Anthropic
- this SDK is a Go native client for the Anthropic API
- this SDK is the one recommeded by Anthropic for Go.
- It's in alpha now but seems to work without problems for these use cases.
- Environment Variables Required:
    - export ANTHROPIC_API_KEY=

Gemini

- Gemini
- this is the official Go client for the Gemini API supported by Google
- **The Gemini SDK requires a "GEMINI_API_KEY" environment variable to authenticate**
- Environment Variables REquired:
    - export GEMINI_API_KEY=

OpenAI

- OpenAI
- Uses OpenAI HTTP API directly
- **The OpenAI API HTTP API requires a "OPENAI_API_KEY" environment variable to authenticate**
- **If you connecting to an OpenAI model to a server besides the official OpenAI servers, you will need to set the "OPENAI_BASE_URL" environment variable to the base URL of the server you are connecting to**
- Environment Variables REquired:
    - export OPENAI_API_KEY=
    - export OPENAI_BASE_URL=https://api.openai.com/v1/chat/completions

MetaLlama

- Meta-LLAMA

DeepSeek

- DeepSeek
- Uses OpenAI HTTP API directly,
- **The API requires a "DEEPSEEK_API_KEY" and "DEEPSEEK_BASE_URL environment variables to authenticate**
- Deepseek was tested using the LLAMA API Provider